Security Issues for Cloud Computing

Kevin Hamlen, The University of Texas at Dallas, USA

Murat Kantarcioglu, The University of Texas at Dallas, USA

Latifur Khan, The University of Texas at Dallas, USA

Bhavani Thuraisingham, The University of Texas at Dallas, USA

ABSTRACT

In this paper, the authors discuss security issues for cloud computing and present a layered framework for secure clouds and then focus on two of the layers, i.e., the storage layer and the data layer. In particular, the authors discuss a scheme for secure third party publications of documents in a cloud. Next, the paper will converse secure federated query processing with map Reduce and Hadoop, and discuss the use of secure co-processors for cloud computing. Finally, the authors discuss XACML implementation for Hadoop and discuss their beliefs that building trusted applications from untrusted components will be a major aspect of secure cloud computing.

Keywords: *Access Control, Authentication, Bottom-Up Design, Data Mining, Information Processing, Ontology Theory, Real-Time IS*

INTRODUCTION

There is a critical need to securely store, manage, share and analyze massive amounts of complex (e.g., semi-structured and unstructured) data to determine patterns and trends in order to improve the quality of healthcare, better safeguard the nation and explore alternative energy. Because of the critical nature of the applications, it is important that clouds be secure. The major security challenge with clouds is that the owner of the data may not have control of where the data is placed. This is because if one wants to exploit the benefits of using cloud computing, one must also utilize the resource allocation and scheduling provided by clouds. Therefore, we need to safeguard the data in the midst of untrusted processes.

The emerging cloud computing model attempts to address the explosive growth of web-connected devices, and handle massive amounts of data. Google has now introduced the MapReduce framework for processing large amounts of data on commodity hardware. Apache's Hadoop distributed file system (HDFS) is emerging as a superior software component for cloud computing combined with integrated parts such as MapReduce. The

DOI: 10.4018/jisp.2010040103

need to augment human reasoning, interpreting, and decision-making abilities has resulted in the emergence of the Semantic Web, which is an initiative that attempts to transform the web from its current, merely human-readable form, to a machine-processable form. This in turn has resulted in numerous social networking sites with massive amounts of data to be shared and managed. Therefore, we urgently need a system that can scale to handle a large number of sites and process massive amounts of data. However, state of the art systems utilizing HDFS and MapReduce are not sufficient due to the fact that they do not provide adequate security mechanisms to protect sensitive data.

We are conducting research on secure cloud computing. Due to the extensive complexity of the cloud, we contend that it will be difficult to provide a holistic solution to securing the cloud, at present. Therefore, our goal is to make increment enhancements to securing the cloud that will ultimately result in a secure cloud. In particular, we are developing a secure cloud consisting of hardware (includes 800TB of data storage on a mechanical disk drive, 2400 GB of memory and several commodity computers), software (includes Hadoop) and data (includes a semantic web data repository). Our cloud system will: (a) support efficient storage of encrypted sensitive data, (b) store, manage and query massive amounts of data, (c) support fine-grained access control and (d) support strong authentication. This paper describes our approach to securing the cloud. The organization of this paper is as follows: In section 2, we will give an overview of security issues for cloud. In section 3, we will discuss secure third party publication of data in clouds. In section 4, we will discuss how encrypted data may be queried. Section 5 will discuss Hadoop for cloud computing and our approach to secure query processes with Hadoop. The paper is concluded in section 6.

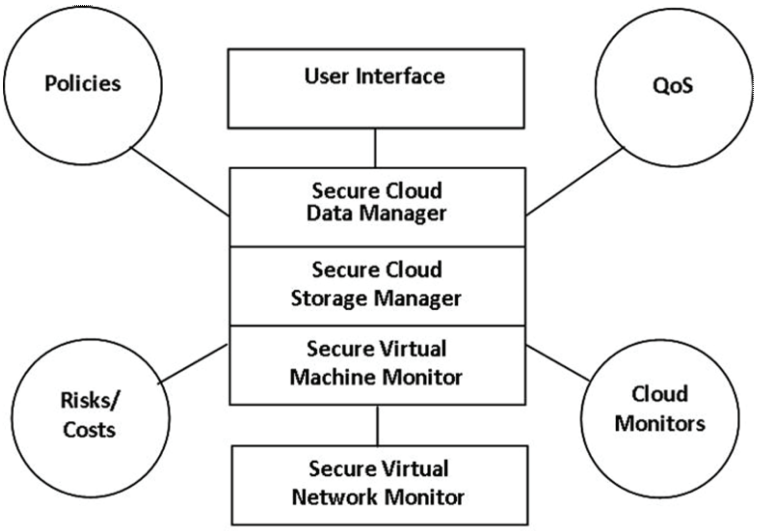
SECURITY ISSUES FOR CLOUDS

There are numerous security issues for cloud computing as it encompasses many technologies including networks, databases, operating systems, virtualization, resource scheduling, transaction management, load balancing, concurrency control and memory management. Therefore, security issues for many of these systems and technologies are applicable to cloud computing. For example, the network that interconnects the systems in a cloud has to be secure. Furthermore, virtualization paradigm in cloud computing results in several security concerns. For example, mapping the virtual machines to the physical machines has to be carried out securely. Data security involves encrypting the data as well as ensuring that appropriate policies are enforced for data sharing. In addition, resource allocation and memory management algorithms have to be secure. Finally, data mining techniques may be applicable to malware detection in clouds.

We have extended the technologies and concepts we have developed for secure grid to a secure cloud. We have defined a layered framework for assured cloud computing consisting of the secure virtual machine layer, secure cloud storage layer, secure cloud data layer, and the secure virtual network monitor layer (Figure 1). Cross cutting services are provided by the policy layer, the cloud monitoring layer, the reliability layer and the risk analysis layer.

For the Secure Virtual Machine (VM) Monitor we are combining both hardware and software solutions in virtual machines to handle problems such as key logger examining XEN developed at the University of Cambridge and exploring security to meet the needs of our applications (e.g., secure distributed storage and data management). For Secure Cloud Storage Management, we are developing a storage infrastructure which integrates resources from multiple providers to form a massive virtual storage system. When a storage node hosts the data from multiple domains, a VM will be cre-

Figure 1. Layered framework for assured cloud

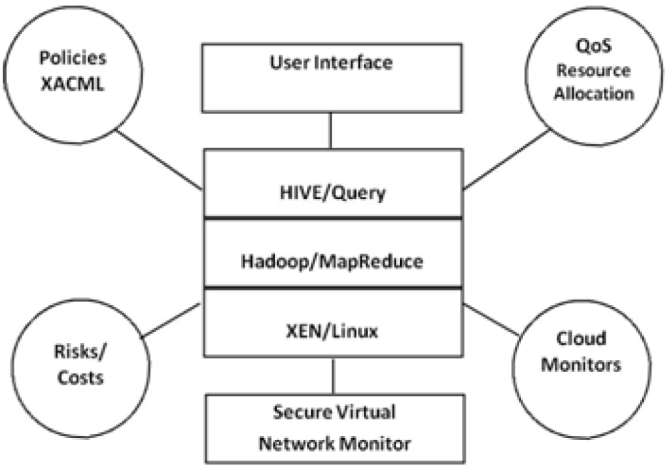


ated for each domain to isolate the information and corresponding data processing. Since data may be dynamically created and allocated to storage nodes, we are investigating secure VM management services including VM pool management, VM diversification management, and VM access control management. Hadoop and MapReduce are the technologies being used. For Secure Cloud Data Management, we have developed secure query processing algorithms for RDF (Resource Description Framework) and SQL (HIVE) data in clouds with an XACML-based (eXtensible Access Control Markup Language) policy manager utilizing the Hadoop/MapReduce Framework. For Secure Cloud Network Management, our goal is to implement a Secure Virtual Network Monitor (VNM) that will create end-to-end virtual links with the requested bandwidth, as well as monitor the computing resources. Figure 2 illustrates the technologies we are utilizing for each of the layers.

This project is being carried out in close collaboration with the AFOSR MURI project on Assured Information Sharing and EOARD funded research project on policy management for information sharing. We have completed a robust demonstration of secure query processing. We have also developed secure storage algorithms and completed the design of XACML for Hadoop. Since Yahoo has come up with a secure Hadoop, we can now implement our design. We have also developed access control and accountability for cloud.

In this paper, we will focus only on some aspects of the secure cloud, namely aspects of the cloud storage and data layers. In particular, (i) we describe ways of efficiently storing the data in foreign machines, (ii) querying encrypted data, as much of the data on the cloud may be encrypted and (iii) secure query processing of the data. We are using Hadoop distributed file system for virtualization at the storage level and applying security for Hadoop which includes an XACML implementation. In addition, we are investigating secure federated query processing on clouds over Hadoop. These efforts will be described in the subsequent sections. Subsequent papers will describe the design and implementation of each of the layers.

Figure 2. Layered framework for assured cloud



THIRD PARTY SECURE DATA PUBLICATION APPLIED TO CLOUD

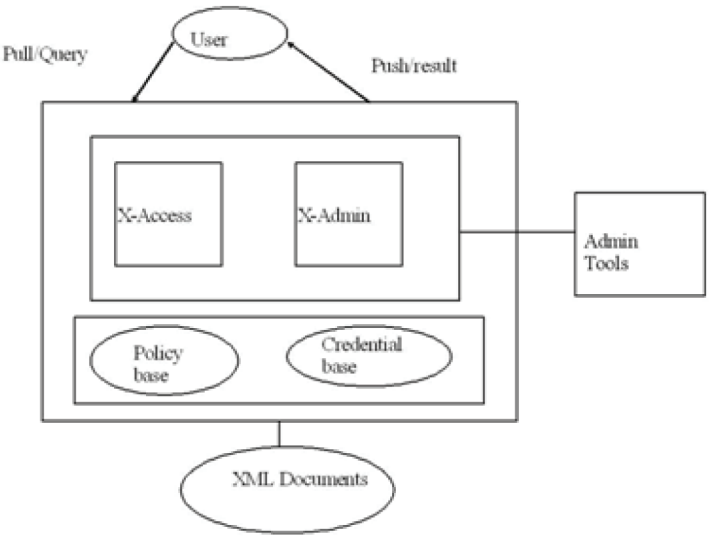
Cloud computing facilitates storage of data at a remote site to maximize resource utilization. As a result, it is critical that this data be protected and only given to authorized individuals. This essentially amounts to secure third party publication of data that is necessary for data outsourcing, as well as external publications. We have developed techniques for third party publication of data in a secure manner. We assume that the data is represented as an XML document. This is a valid assumption as many of the documents on the web are now represented as XML documents. First, we discuss the access control framework proposed in Bertino (2002) and then discuss secure third party publication discussed in Bertino (2004).

In the access control framework proposed in Bertino (2002), security policy is specified depending on user roles and credentials (see Figure 3). Users must possess the credentials to access XML documents. The credentials depend on their roles. For example, a professor has access to all of the details of students while a secretary only has access to administrative information. XML specifications are used to specify the security policies. Access is granted for an entire XML document or portions of the document. Under certain conditions, access control may be propagated down the XML tree.

For example, if access is granted to the root, it does not necessarily mean access is granted to all the children. One may grant access to the XML schema and not to the document instances. One may grant access to certain portions of the document. For example, a professor does not have access to the medical information of students while he has access to student grade and academic information. Design of a system for enforcing access control policies is also described in Bertino (2002). Essentially, the goal is to use a form of view modification so that the user is authorized to see the XML views as specified by the policies. More research needs to be done on role-based access control for XML and the semantic web.

In Bertino (2004), we discuss the secure publication of XML documents (see Figure 4). The idea is to have untrusted third party publishers. The owner of a document specifies access control polices for the subjects. Subjects get the policies from the owner when they subscribe to a document. The owner sends the documents to the Publisher. When the subject requests a document, the publisher will apply the policies

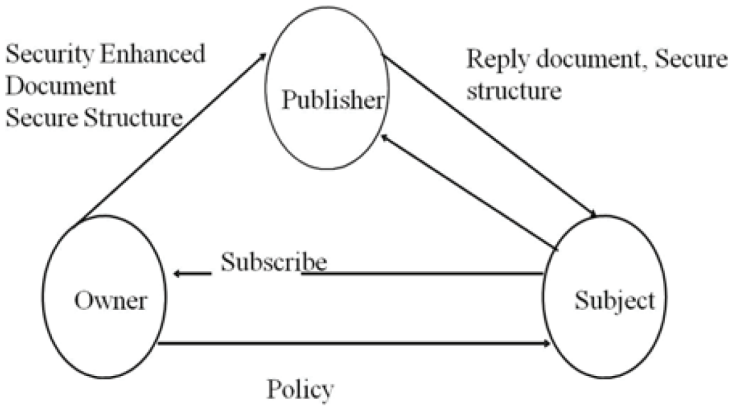
Figure 3. Access control framework



relevant to the subject and give portions of the documents to the subject. Now, since the publisher is untrusted, it may give false information to the subject. Therefore, the owner will encrypt various combinations of documents and policies with his/her private key. Using Merkle signature and the encryption techniques, the subject can verify the authenticity and completeness of the document (see Figure 4 for secure publishing of XML documents).

In the cloud environment, the third party publisher is the machine that stored the sensitive data in the cloud. This data has to be protected and the techniques we have discussed above have to be applied to that authenticity and completeness can be maintained.

Figure 4. Secure third party publication



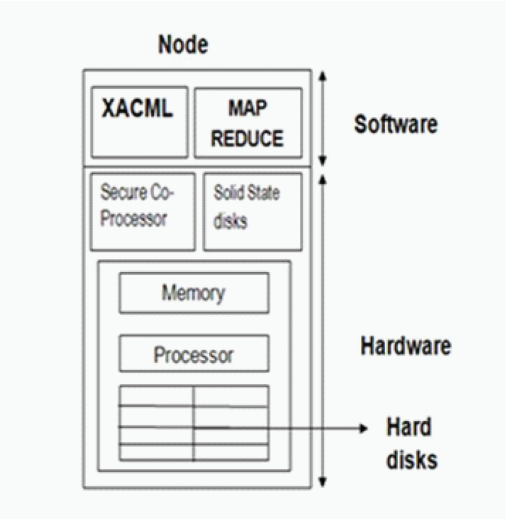
ENCRYPTED DATA STORAGE FOR CLOUD

Since data in the cloud will be placed anywhere, it is important that the data is encrypted. We are using secure co-processor as part of the cloud infrastructure to enable efficient encrypted storage of sensitive data. One could ask us the question: why not implement your software on hardware provided by current cloud computing systems such as Open Cirrus? We have explored this option. First, Open Cirrus provides limited access based on their economic model (e.g., Virtual cash). Furthermore, Open Cirrus does not provide the hardware support we need (e.g., secure co-processors). By embedding a secure co-processor (SCP) into the cloud infrastructure, the system can handle encrypted data efficiently (see Figure 5).

Basically, SCP is a tamper-resistant hardware capable of limited general-purpose computation. For example, IBM 4758 Cryptographic Coprocessor (IBM) is a single-board computer consisting of a CPU, memory and special-purpose cryptographic hardware contained in a tamper-resistant shell, certified to level 4 under FIPS PUB 140-1. When installed on the server, it is capable of performing local computations that are completely hidden from the server. If tampering is detected, then the secure co-processor clears the internal memory. Since the secure coprocessor is tamper-resistant, one could be tempted to run the entire sensitive data storage server on the secure co-processor. Pushing the entire data storage functionality into a secure co-processor is not feasible due to many reasons.

First of all, due to the tamper-resistant shell, secure co-processors have usually limited memory (only a few megabytes of RAM and a few kilobytes of non-volatile memory) and computational power (Smith, 1999). Performance will improve over time, but problems such as heat dissipation/power use (which must be controlled to avoid disclosing processing) will force a gap between general purposes and secure computing. Another issue is that the software running on the SCP must be totally trusted and verified. This security requirement implies that the software running on the SCP should be kept as simple as possible. So how does this hardware help in storing large sensitive data sets? We can encrypt the sensitive data sets using random private keys and to alleviate the risk of key disclosure, we can use tamper-resistant hardware to store some of the

Figure 5. Parts of the proposed instrument



encryption/decryption keys (i.e., a master key that encrypts all other keys). Since the keys will not reside in memory unencrypted at any time, an attacker cannot learn the keys by taking the snapshot of the system. Also, any attempt by the attacker to take control of (or tamper with) the co-processor, either through software or physically, will clear the co-processor, thus eliminating a way to decrypt any sensitive information. This framework will facilitate (a) secure data storage and (b) assured information sharing. For example, SCPs can be used for privacy preserving information integration which is important for assured information sharing.

We have conducted research on querying encrypted data as well as secure multipart computation (SMC). With SMC protocols, one knows about his own data but not his partner's data since the data is encrypted. However, operations can be performed on the encrypted data and the results of the operations are available for everyone, say, in the coalition to see. One drawback of SMC is the high computation costs. However, we are investigating more efficient ways to develop SMC algorithms and how these mechanisms can be applied to a cloud.

SECURE QUERY PROCESSING WITH HADOOP

Overview of Hadoop

A major part of our system is HDFS which is a distributed Java-based file system with the capacity to handle a large number of nodes storing petabytes of data. Ideally a file size is a multiple of 64 MB. Reliability is achieved by replicating the data across several hosts. The default replication value is 3 (i.e., data is stored on three nodes). Two of these nodes reside on the same rack while the other is on a different rack. A cluster of data nodes constructs the file system. The nodes transmit data over HTTP and clients' access data using a web browser. Data nodes communicate with each other to regulate, transfer and replicate data.

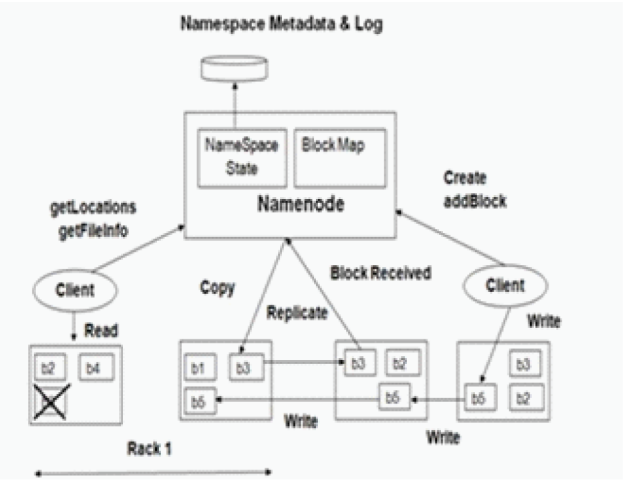
HDFS architecture is based on the Master-Slave approach (Figure 6). The master is called a Namenode and contains metadata. It keeps the directory tree of all files and tracks which data is available from which node across the cluster. This information is stored as an image in memory. Data blocks are stored in Datanodes. The namenode is the single point of failure as it contains the metadata. So, there is optional secondary Namenode that can be setup on any machine. The client accesses the Namenode to get the metadata of the required file. After getting the metadata, the client directly talks to the respective Datanodes in order to get data or to perform IO actions (Hadoop). On top of the file systems there exists the *map/reduce engine*. This engine consists of a Job Tracker. The client applications submit map/reduce jobs to this engine. The Job Tracker attempts to place the work near the data by pushing the work out to the available Task Tracker nodes in the cluster.

Inadequacies of Hadoop

Current systems utilizing Hadoop have the following limitations:

(1) **No facility to handle encrypted sensitive data:** Sensitive data ranging from medical records to credit card transactions need to be stored using encryption techniques for additional protection. Currently, HDFS does not perform secure and efficient query processing over encrypted data.
(2) **Semantic Web Data Management**: There is a need for viable solutions to improve the performance and scalability of queries against semantic web data such as RDF (Resource Description Framework). The number of RDF datasets is increasing. The problem of storing billions of RDF triples and the ability to efficiently query them is yet to be solved (Muys, 2006; Teswanich, 2007; Ramanujam, 2009). At present, there is no support to store and retrieve RDF data in HDFS.
(3) **No fine-grained access control:** HDFS does not provide fine-grained access control. There is some work to provide access control lists for HDFS (Zhang, 2009). For

Figure 6. Hadoop Distributed File System (HDFS architecture)



many applications such as assured information sharing, access control lists are not sufficient and there is a need to support more complex policies.

(4) **No strong authentication:** A user who can connect to the JobTracker can submit any job with the privileges of the account used to set up the HDFS. Future versions of HDFS will support network authentication protocols like Kerberos for user authentication and encryption of data transfers (Zhang, 2009). However, for some assured information sharing scenarios, we will need public key infrastructures (PKI) to provide digital signature support.

System Design

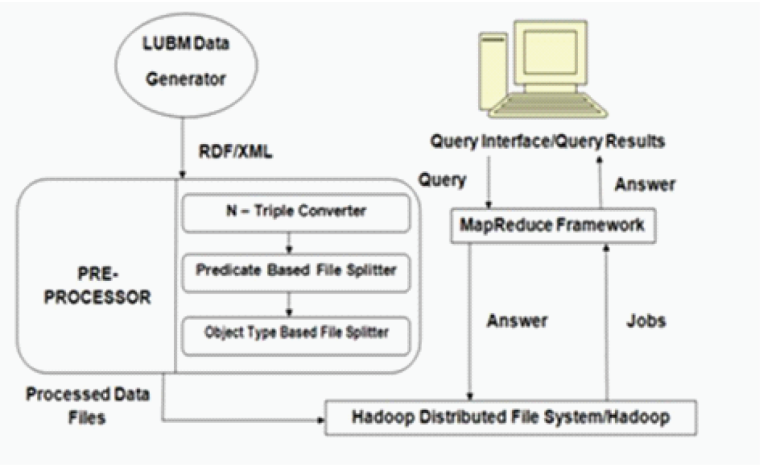
While the secure co-processors can provide the hardware support to query and store the data, we need to develop a software system to store, query, and mine the data. More and more applications are now using semantic web data such as XML and RDF due to their representation power especially for web data management. Therefore, we are exploring ways to securely query semantic web data such as RDF data on the cloud. We are using several software tools that are available to help us in the process including the following:

Jena: Jena is a framework which is widely used for solving SPARQL queries over RDF data (Jena). But the main problem with Jena is scalability. It scales in proportion to the size of main-memory. It does not have distributed processing. However, we will be using Jena in the initial stages of our preprocessing steps.

Pellet: We use Pellet to reason at various stages. We do real-time query reasoning using pellet libraries (Pellet) coupled with Hadoop's map-reduce functionalities.

Pig Latin: Pig Latin is a scripting language which runs on top of Hadoop (Gates, 2009). Pig is a platform for analyzing large data sets. Pig's language, Pig Latin, facilitates sequence of data transformations such as merging data sets, filtering them, and applying functions to records or groups of records. It comes with many built-in functions, but we can also create our own user-defined functions to do special-purpose processing. Using this scripting language, we will avoid writing our own map-reduce code; we will rely on Pig Latin's scripting power that

Figure 7. System architecture for SPARQL query optimization



will automatically generate script code to map-reduce code.

Mahout, Hama: These are open source data mining and machine learning packages that already augment Hadoop (Mahout) (Hama) (Moretti, 2008).

Our approach consists of processing SPARQL queries securely over Hadoop. SPARQL is a query language used to query RDF data (W3C, *SPARQL*). The software part we will develop is a framework to query RDF data distributed over Hadoop (Newman, 2008; McNabb, 2007). There are a number of steps to preprocess and query RDF data (see Figure 7). With this proposed part, researchers can obtain results to optimize query processing of massive amounts of data. Below we discuss the steps involved in the development of this part.

Pre-processing: Generally, RDF data is in XML format (see Lehigh University Benchmark [LUBM] RDF data). In order to execute a SPARQL query, we propose some data pre-processing steps and store the pre-processed data into HDFS. We have an N-triple Convertor module which converts RDF/XML format of data into N-triple format as this format is more understandable. We will use Jena framework as stated earlier, for this conversion purpose. In Predicate Based File Splitter module, we split all N-triple format files based on the predicates. Therefore, the total number of files for a dataset is equal to the number of predicates in the ontology/taxonomy. In the last module of the pre-processing step, we further divide predicate files on the basis of the type of object it contains. So, now each predicate file has specific types of objects in it. This is done with the help of the Pellet library. This pre-processed data is stored into Hadoop.

Query Execution and Optimization: We are developing a SPARQL query execution and optimization module for Hadoop. As our storage strategy is based on predicate splits, first, we will look at the predicates present in the query. Second, rather than looking at all of the input files, we will look at a subset of the input files that are matched with predicates. Third, SPARQL queries generally have many joins in them and all of these joins may not be possible to perform in a single Hadoop job. Therefore, we will devise an algorithm that decides the number of jobs required for each kind of query. As part of optimiza-

tion, we will apply a greedy strategy and cost-based optimization to reduce query processing time. An example of greedy strategy is to cover the maximum number of possible joins in a single job. For the cost model, the join to be performed first is based on summary statistics (e.g., selectivity factor of a bounded variable, join triple selectivity factor for three triple patterns. For example, consider a query for LUBM dataset: "List all persons who are alumni of a particular university." In SPARQL:

```
PREFIX rdf: <http://www.
w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.
edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X WHERE {
?X rdf:type ub:Person .
<http://www.University0.edu>
ub:hasAlumnus ?X }
```

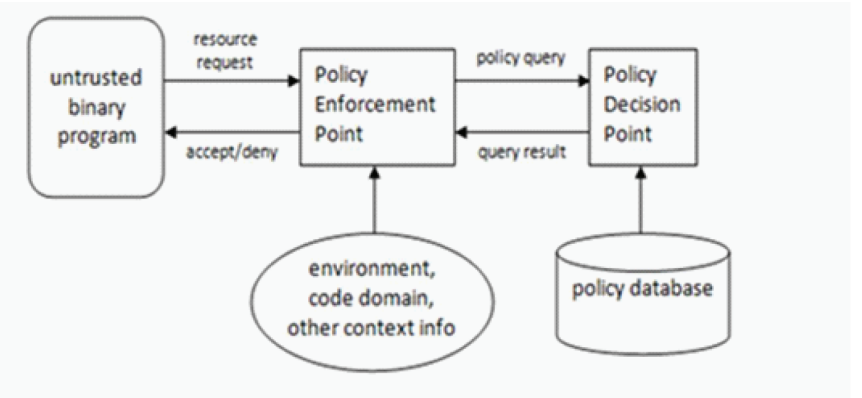
The query optimizer will take this query input and decide a subset of input files to look at based on predicates that appear in the query. Ontology and pellet reasoner will identify three input files (underGraduateDegreeFrom, masterDegreeFrom and DoctoraldegreeFrom) related to predicate, "hasAlumns". Next, from type file we filter all the records whose objects are a subclass of Person using the pellet library. From these three input files (underGraduateDegreeFrom, masterDegreeFrom and DoctoraldegreeFrom) the optimizer filters out triples on the basis of <http://www.University0.edu> as required in the query. Finally, the optimizer determines the requirement for a single job for this type of query and then the join is carried out on the variable X in that job.

With respect to secure query processing, we are investigating two approaches. One is rewriting the query in such a way that the policies are enforced in an appropriate manner. The second is query modification where the policies are used in the "where" clause to modify the query.

Integrate SUN XACML Implementation into HDFS

Current Hadoop implementations enforce a very coarse-grained access control policy that permits or denies a principal access to essentially all system resources as a group without distinguishing amongst resources. For example, users who are granted access to the Namenode (see Figure 6) may execute any program on any client machine, and all client machines have read and write access to all files stored on all clients. Such coarse-grained security is clearly unacceptable when data, queries, and the system resources that implement them are security-relevant, and when not all users and processes are fully trusted. Current work (Zhang, 2009) addresses this by implementing standard access control lists for Hadoop to constrain access to certain system resources, such as files; however, this approach has the limitation that the enforced security policy is baked into the operating system and therefore cannot be easily changed without modifying the operating system. We are enforcing more flexible and fine-grained access control policies on Hadoop by designing an In-lined Reference Monitor implementation of Sun XACML. XACML (Moses, 2005) is an OASIS standard for expressing a rich language of access control policies in XML. Subjects, objects, relations, and contexts are all generic and extensible in XACML, making it well-suited for a distributed environment where many different sub-policies may interact to form larger, composite, system-level policies. An abstract XACML enforcement mechanism is depicted in Figure 8. Untrusted processes in the framework access security-relevant resources by submitting a request to the resource's Policy Enforcement Point (PEP). The PEP reformulates the request as a policy query and submits it to a Policy Decision Point (PDP). The PDP consults any policies related to the request to answer the query. The PEP either grants or denies the resource request based on the answer it receives. While the PEP and PDP components of the enforcement mechanism are traditionally implemented at

Figure 8. XACML enforcement architecture



the level of the operating system or as trusted system libraries, we propose to achieve greater flexibility by implementing them in our system as In-lined Reference Monitors (IRM's). IRM's implement runtime security checks by in-lining those checks directly into the binary code of untrusted processes. This has the advantage that the policy can be enforced without modifying the operating system or system libraries. IRM policies can additionally constrain program operations that might be difficult or impossible to intercept at the operating system level. For example, memory allocations in Java are implemented as Java bytecode instructions that do not call any external program or library. Enforcing a fine-grained memory-bound policy as a traditional reference monitor in Java therefore requires modifying the Java virtual machine or JIT-compiler. In contrast, an IRM can identify these security-relevant instructions and inject appropriate guards directly into the untrusted code to enforce the policy.

Finally, IRM's can efficiently enforce history-based security policies, rather than merely policies that constrain individual security-relevant events. For example, our past work (Jones, 2009) has used IRMs to enforce fairness policies that require untrusted applications to share as much data as they request. This prevents processes from effecting denial of service attacks based on freeloading behavior. The code injected into the untrusted binary by the IRM constrains each program operation based on the past history of program operations rather than in isolation. This involves injecting security state variables and counters into the untrusted code, which is difficult to accomplish efficiently at the operating system level (Figure 8).

The core of an IRM framework consists of a *binary-rewriter*, which statically modifies the binary code of each untrusted process before it is executed to insert security guards around potentially dangerous operations. Our proposed binary-rewriter implementation will be based on SPoX (Security Policy XML) (Hamlen, 2008), which we developed to enforce declarative, XML-based, IRM policies for Java bytecode programs. In order to provide strong security guarantees for our system, we will apply automated software verification technologies, including type and model-checking, which we have previously used to certify the output of binary-rewriters (Hamlen, 2006; DeVries, 2009). Such certification allows a small, trusted verifier to independently prove that rewritten binary code satisfies the original security policy, thereby shifting the comparatively larger binary-rewriter out of the trusted computing base of the system.

Strong Authentication

Currently, Hadoop does not authenticate users. This makes it hard to enforce access control

for security sensitive applications and makes it easier for malicious users to circumvent file permission checking done by HDFS. To address these issues, the open source community is actively working to integrate Kerberos protocols with Hadoop (Zhang, 2009). On top of the proposed Kerboros protocol, for some assured information applications, there may be a need for adding simple authentication protocols to authenticate with secure co-processors. For this reason, we can add a simple public key infrastructure to our system so that users can independently authenticate with secure co-processors to retrieve secret keys used for encrypting sensitive data. We can use open source public key infrastructure such as the OpenCA PKI implementation for our system (OpenCA).

SUMMARY AND CONCLUSION

In this paper, we first discussed security issues for cloud. These issues include storage security, middleware security, data security, network security and application security. The main goal is to securely store and manage data that is not controlled by the owner of the data. Then we focused on specific aspects of cloud computing. In particular, we are taking a bottom up approach to security where we are working on small problems in the cloud that we hope will solve the larger problem of cloud security. First, we discussed how we may secure documents that may be published in a third party environment. Next, we discussed how secure co-processors may be used to enhance security. Finally, we discussed how XACML may be implemented in the Hadoop environment as well as in secure federated query processing with SPARQL using MapReduce and Hadoop.

There are several other security challenges including security aspects of virtualization. We believe that due to the complexity of the cloud, it will be difficult to achieve end-to-end security. However, the challenge we have is to ensure more secure operations even if some parts of the cloud fail. For many applications, we not only need information assurance but also mission assurance. Therefore, even if an adversary has entered the system, the objective is to thwart the adversary so that the enterprise has time to carry out the mission. As such, building trust applications from untrusted components will be a major aspect with respect to cloud security.

REFERENCES

W3C. (n.d.). *SPARQL*. Retrieved from http://www.w3.org/TR/rdf-sparql-query

Bertino, E. (2002). Access Control for XML Documents. *Data & Knowledge Engineering*, *43*(3).

Bertino, E. (2004). Selective and Authentic Third Party Distribution of XML Documents. *IEEE Transactions on Knowledge and Data Engineering*, *16*(10). doi:10.1109/TKDE.2004.63

DeVries, B. W., Gupta, G., Hamlen, K. W., Moore, S., & Sridhar, M. (2009). ActionScript Bytecode Verification with Co-Logic Programming. In *Proceedings of the ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS)*.

Gates, F., Natkovich, O., Chopra, S., Kamath, S. M., Kamath, P., Narayanamuthry, S. M., et al. (2009). Building a High-Level Dataflow System on top of Map-Reduce: The Pig Experience. In *Proceedings of the Thirty-Fifth International Conference on Very Large Data Bases (VLDB) (Industrial, Applications and Experience Track)*, Lyon, France.

Hadoop. (n.d.). Retrieved from http://hadoop.apache.org

Hama. (n.d.). Retrieved from http://cwiki.apache.org/labs/cloudsglossary.html

Hamlen, K. W., & Jones, M. (2008). Aspect-Oriented In-lined Reference Monitors. In *Proceedings of the ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS)*.

Hamlen, K. W., Morrisett, G., & Schneider, F. B. (2006). Certified In-lined Reference Monitoring on. NET. In *Proceedings of the ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS)*.

IBM. (2004). *IBM PCI cryptographic coprocessor*. Retrieved from http://www.ibm.com/security/cryptocards.

Jena. (n.d.). Retrieved from http://jena.sourceforge.net

Jones, M., & Hamlen, K. W. (2009). Enforcing IRM Security Policies: Two Case Studies. In *Proceedings of the IEEE Intelligence and Security Informatics Conference (ISI)*.

Lehigh University Benchmark (LUBM). (n.d.). Retrieved from http://swat.cse.lehigh.edu/projects/lubm

Mahout. (n.d.). Retrieved from http://lucene.apache.org/mahout/

McNabb, A., Monson, C., & Seppi, K. (2007). MRPSO: MapReduce particle swarm optimization. In *Proceedings of the 9th annual conference on genetic and evolutionary computation (GECCO 2007)* (p. 177). New York: ACM Press.

Moretti, C., Steinhaeuser, K., Thainer, D., & Chawla, N. (2008). Scaling Up Classifiers to Cloud Computers. In *Proceedings of the IEEE ICDM*.

Moses, T. (Ed.). (2005, February). eXtensible Access Control Markup Language (XACML) Version 2.0. *OASIS Standard*. Retrieved from http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

Muys, A. (2006). *Building an Enterprise-Scale Database for RDF Data*.

Newman, A., Hunter, J., Li, Y. F., Bouton, C., & Davis, M. (2008). In *Proceedings of a Scale-Out RDF Molecule Store for Distributed Processing of Biomedical Data Semantic Web for Health Care and Life Sciences Workshop (WWW 2008)*.

Open, C. A. *Implementation*. (n.d.). Retrieved from http://www.openca.org/projects/openca/

Pellet. (n.d.). Retrieved from http://clarkparsia.com/pellet

Ramanujam, S., Gupta, A., Khan, L., & Seida, S. (2009). R2D: Extracting relational structure from RDF stores. In *Proceedings of the ACM/IEEE International Conference on Web Intelligence,* Milan, Italy.

Smith, S., & Weingart, S. (1999). Building a high-performance, programmable secure coprocessor. [Special Issue on Computer Network Security]. *Computer Networks*, *31*, 831–860. doi:10.1016/S1389-1286(98)00019-X

Teswanich, W., & Chittayasothorn, S. (2007). A Transformation of RDF Documents and Schemas to Relational Databases. *IEEE Pacific Rim Conferences on Communications, Computers, and Signal Processing*, 38-41.

Zhang, K. (2009). *Adding user and service-to-service authentication to Hadoop*. Retrieved from https://issues.apache.org/jira/browse/HADOOP-4343